60-100 LECTRURE 25
REVISION – This is the Final Exam for Fall 2007
Answer ALL ten questions. SHOWALL WORK to obtain marks. Put a box around your final answer. Answer in ink if you wish to review your paper after it has been marked.

Note that the answers will be developed in the lecture

The answers are also available on the 60-100 course website – follow the link

"Answers to previous final exams

Q1)

a) Write a Miranda program `p1a` which takes a list of pairs of numbers as input and which returns the sum of each pair's product. You must use `foldr`, `map` and composition `(.)` in your answer. The following is an example of the use of `p1a`:

```
p1a [(3,4), (2, 5), (3,1)] => 25
```

b) Use a list comprehension to define a program `p1b` which takes a list of numbers as input, and which returns a list of squares of all those numbers in the input that are greater than `5`. For example:

```
p1b [3, 6, 4, 7, 8] => [36, 49, 64]
```

Q2)

a) Consider the following program:

```
p2 [] = 4
p2 [x,y] = 3
p2 (x:xs) = p2 [1,2] + p2 xs
```

What is the result of the application:

```
p2 [3,4,5,6,7]
```

b) What is the type of the following data item :

```
([], [(3,'a'),(4,'s')]:[])
```

Q3) What is the type of the following program `p3`:

```
p3 a b c = [c], if a!2 = 's'
         = b!0, otherwise
```

Q4)

a)Write a Miranda program to implement the following relational algebra operator:

`select_second_of_three`

b) Write a Miranda program to implement the following relational algebra operator:

`join_first_of_two_with_secnd_of_three`

Q5 Use RECURSION to write a Miranda program `p5` which takes two lists as input and which zips up the lists into a single list of pairs. For example:

```
p5 [4,5,2,7] [`f',`h',`s']

      =>[(4,'f'),(5,'h',),(2,'s')]
```

Note that if the two lists are not of the same length, then the surplus elements in the longer list are discarded.

Hint: recurse on both lists at the same time.

Q6) Let the grammar G be defined as follows:

```
 G =
(terminals = [0,1,2,3,-,+,*,^, (, ) }
 non-terminals = {dig, op, expr}
 start symbol = expr
 productions =
     { expr ::= num
              | expr op expr
              | -expr
              | ( expr )
        dig ::= 0 | 1 | 2 | 3
         op ::= + | * | ^ })
```

Draw a syntax tree for the following expression:

```
- ( 2 ^ ( 3 + 4))
```

Q7) Attribute grammars not covered in Fall 2008

Q8) Use structural induction on the length of the first list to prove that the length of the output from the program `p8` is equal to the sum of the lengths of its two input lists:

```
p8    []    ys = ys
p8 (x:xs) ys = x (p8 xs ys)
```

That is, prove: `#(p8 n m) = #n + #m`

Q9) Calculate and illustrate on a graph the complexity of the program `p9` with respect to the length of the input list:

```
p9 [y]     = [y]
p9 (x:xs) = p9 [x]++(x:xs)++ p9 xs
```

Q10)

a) Use a truth table to determine if the following formula is satisfiable, universally valid, or unsatisfiable, where ~ is logical negation:

```
(x -> y) -> (z -> (~x \/ (y & z)))
```

```
x  y  z   (x->y) -> (z->(~x \/ (y & z)))
```

b) Use a truth table to show that the formula

F = (x -> z)

is or is not a logical consequence of the set of formulas

S = {(y \/ x), (~z -> ~y)}

x	y	z	(y\/x)	(~z->~y)	(x->z)